JSF Abstraction Layer Vision

Mark Weinberg

8/3/2004

Goals

- Reduce the lifetime Operating and Support costs of FPGA based signal processing on JSF
  - Enable cost effective selection of HW on JSF
  - Minimize rework required as HW is changed
    - Assure that only the abstraction layer may need rework as HW is changed
- Reduce FPGA based application development costs
  - Provide stable, well-abstracted interface to HW resources
  - Provision for improvements in HW which will occur over the JSF lifetime (30 years)
  - Provision for application growth areas

FPGA-Based Signal Processing Challenges

- Anticipated evolution of COTS and Applications require that SW and HW be refreshed on a 3 year life-cycle for the next 30 years
  - A cost effective development platform for FPGA based signal processing is imperative
- FPGA based Signal Processing generally requires major rework with only small changes to the HW platform
- Major changes to the HW configuration usually require application re-design from scratch.
- FPGA resources are very limited, so an abstraction layer is usually resource prohibitive
- Without an abstraction layer or good systems analysis tools, modular design of applications is impossible

The Solution

- Develop a toolset which includes
  - Levels of abstraction a la the software model:
    - Application/OS/BSP(or BIOS)
      - Affords minor changes of HW (e.g. changing memory chips) with modification to only the lowest layer
      - Affords major HW changes (e.g. changing from RACE++ to RapidIO) with only modification to the lowest two layers
      - Provides application developers with a well-defined, stable interface
  - A self-scaling abstraction layer
    - Abstraction layer includes only the components that it needs in the HW instantiation
      - Efficient use of limited FPGA resources
  - A path from systems analysis of a given application-to-HW-mapping to instantiation of that mapping
    - Mappings are analyzed with respect to meeting deadlines on communications and data transactions.
    - A mechanism is provided to automate mapping to instantiation
    - Implies that multiple tools work hand-in-hand

Terminology

- Signal processing system
  - A combination of hardware and firmware which performs high speed processing on streams of sensor data.
    - The system has associated performance requirements on all processing components and data flows
- Hardware platform
  - The base hardware set. FPGAs with interfaces to memory, communication paths, etc.
- Processing core element
  - FPGA Circuitry designed by the end user to be integated into a signal processing system
- System model
  - A hardware independent view of the signal processing application. A representation of the process, data transfer, order of operation, and triggering specificaion. Typically this is done using a directed acyclic graph (DAG) and auxiliary data to specify operational requirements
- FPGA abstraction layer
  - FPGA Infrastructure circuitry designed to integrate processing core elements with a specified HW platform per a set of given requirements
    - So, both the circuitry and the knowledge of how to integrate it to meet a set of requirements

Four main components of JSF FPGA based signal processing platform

- Systems Analysis Toolset (SAT)
  - (Future development) High level analysis of signal processing system
  - Provides assurance that core processing and communication can be achieved within specified QoS
  - Aids application developers in determining application to HW mapping
- Core Application Development System
  - VHDL simulation/synthesis for signal processing core elements
  - Provisions for integration with abstraction layer and IFS with all timing constraints met
- Interface Firmware Shell (IFS)
  - Low level FPGA interface to external HW
  - Analagous to Board Support Package software
  - Allows on-board HW to be changed without any change to abstraction layer or core elements
- FPGA Abstraction Layer
  - Checks if desired application to HW mapping is achievable (while meeting specifications on HW)
    - Used by SAT to check legal configurations
  - Builds infrastructure (i.e. scaled MUXes, communication link layers/EDAC, and resource interfaces) as required by HW mapping
  - Integrates infrastructure with core elements
  - Affords major system HW reconfigurations or changes (e.g RACE++ to RapidIO migration) without any changes to core elements
  - Provides a stable, well defined, and flexible basis for signal processing application growth
    - Provides scalability to applications

Derived Requirements for the Abstraction Layer

- Abstraction layer provisions for higher level functions of subsystem HW
  - Processing elements see only virtual streams of data
  - Abstraction layer handles all the particulars of resource interface
    - Busing, handshaking, clocking, link layer function
- Abstraction layer guarantees all timing on its interfaces
- Abstraction layer provides MUXing as required by a given mapping.
  - If no mux is required, no mux is instantiated
  - If an n:1 mux is the requirement and (n+1):1 mux requires more FPGA resources, a n:1 mux is instantiated
- If the Abstraction layer requires resources (e.g. FPGA block RAM) those resources are managed along with resources required by processing elements

Derived Requirements for the Abstraction Layer (cont)

- Notional requirements interface between abstraction layer and processing element
  - Data stream requirements for n streams
    - Data storage (i.e. address space, memory map, semaphores, etc)
    - Data integrity requirement (QoS/Pe_{bit})
    - Signal Interface description
  - Fault Detection/Isolation interface
  - Stream BW/Procesessing BW relationship
    - If the stream speed is improved, does the input stream to the processing element improved?
    - How does this effect the overall improvement of process element's performance?
    - What else needs to be done to help besides providing more capacity at the stream? Increase FPGA clock speeds?

Derived Requirements for the Abstraction Layer (cont)

- Notional requirements interface between abstraction layer and high level systems analysis tool
  - Mapping of process elements to FPGA resources description (from systems analysis tool)
    - Which process elements are to live on which FPGAs
    - Which process elements need access to which resources
    - Specification for BW, latency, and backlog (memory size) requirements for every stream
      - Requires knowledge of particulars of the IFS
      - Requires understanding of impact due to MUXes, Link layers, etc
  - Feedback from the abstraction layer tool to see whether this mapping is viable
  - Feedback from abstraction layer to “profile” application
    - Verify analysis models.

Derived Requirements for the Abstraction Layer (cont)

- Notional requirements interface between abstraction layer and the Core Application Development System
  - An understanding of how to integrate the various code pieces and guarantee timing
  - Must work within practical limits of design synthesis
    - Don’t overload system with timing constraints
  - Abstraction layer must provide the core application development system with simulation models of the interface.
    - Note Infrastructure pieces should be either no-impact to the simulation model or be modeled by additional latency.